🎯 Combination Sum

http://leetcode.com/problems/combination-sum/

`ans.add(list);`

- Adds the same `list` object (a reference) to `ans`.
- In recursion, `list` is modified (elements are added and removed).
- Because all entries in `ans` point to the same list, they all change as the list changes.
- This leads to incorrect or duplicated results.

`ans.add(new ArrayList<>(list));`

- Adds a new copy of the current `list` to `ans`.
- This copy stays unchanged even when recursion modifies the original `list`.
- Ensures each stored combination is correct and independent of future changes.

```
public class Main {
	public static void main(String[] args) {
		int[] coin = { 2, 3, 6, 7 };
		int amount = 7;
		List<Integer> list = new ArrayList<>();
		List<List<Integer>> ans = new ArrayList<>();
		comb(coin, amount, list, 0, ans);
		System.out.println(ans);
	}

	public static void comb(int[] coin, int amount, List<Integer> list, int idx,
List<List<Integer>> ans) {
		if (amount == 0) {
			// System.out.println(ll);
			ans.add(list);
			return;
		}

		for (int i = idx; i < coin.length; i++) {
			if (amount >= coin[i]) {
				List<Integer> tempList = new ArrayList<>(list);
				tempList.add(coin[i]);
				comb(coin, amount - coin[i], tempList, i, ans);
			}
		}
	}

}
```

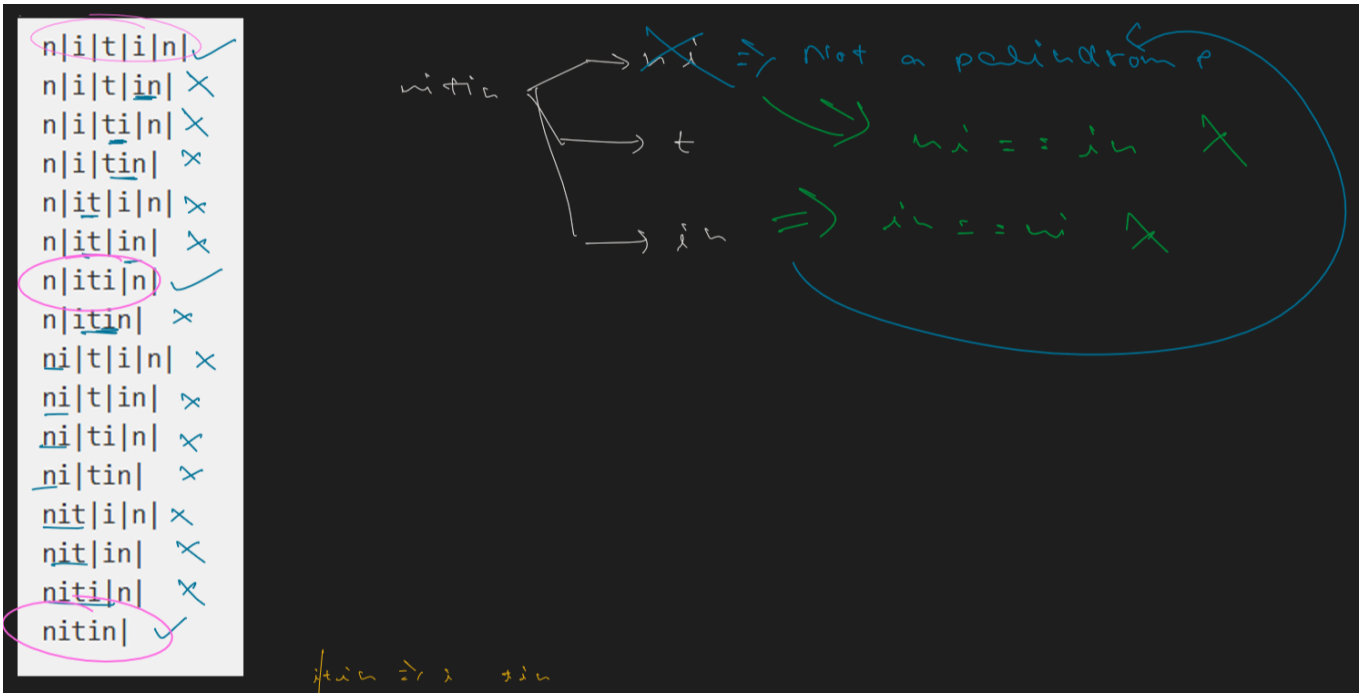
```
public class Main {
	public static void main(String[] args) {
		int[] coin = { 2, 3, 6, 7 };
		int amount = 7;
		List<Integer> list = new ArrayList<>();
		List<List<Integer>> ans = new ArrayList<>();
		comb(coin, amount, list, 0, ans);
```

```
            System.out.println(ans);
      }

      public static void comb(int[] coin, int amount, List<Integer> list, int idx,
List<List<Integer>> ans) {
            if (amount == 0) {
                  // System.out.println(list);
                  ans.add(new ArrayList<>(list));
                  return;
            }

            for (int i = idx; i < coin.length; i++) {
                  if (amount >= coin[i]) {
                     list.add(coin[i]);
                      comb(coin, amount - coin[i], list, i, ans);
                      list.remove(list.size()-1);

                  }
            }
      }

}
```

```
class Solution {
    public List<List<Integer>> combinationSum(int[] candidates, int target) {
        List<Integer> list = new ArrayList<>();
        List<List<Integer>> ans = new ArrayList<>();
        int idx = 0;
        comb(candidates, target, list, idx, ans);
        return ans;
    }

    public static void comb(int[] coin, int amount, List<Integer> list, int idx,
List<List<Integer>> ans) {
        if (amount == 0) {
            // System.out.println(ll);
            ans.add(list);
            return;
        }

        for (int i = idx; i < coin.length; i++) {
            if (amount >= coin[i]) {
                List<Integer> tempList = new ArrayList<>(list);
                tempList.add(coin[i]);
                comb(coin, amount - coin[i], tempList, i, ans);
            }
        }
    }

}
```

🎯 Palindrome Partitioning

https://leetcode.com/problems/palindrome-partitioning/



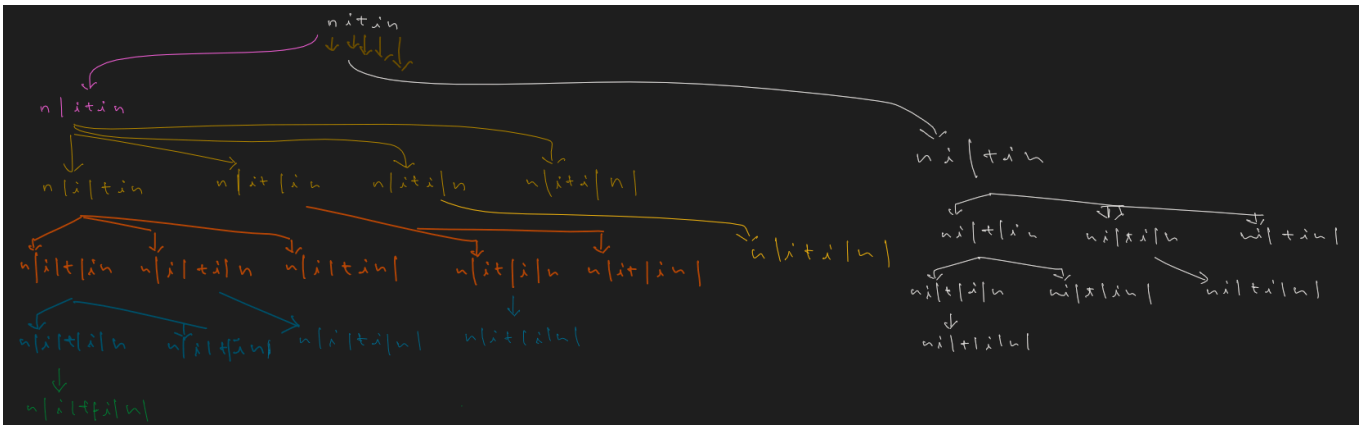
Suppose I have a string **"nitin"**. I want to split it into paths, I have 5 options— essentially, I'll try to split it at every possible position.

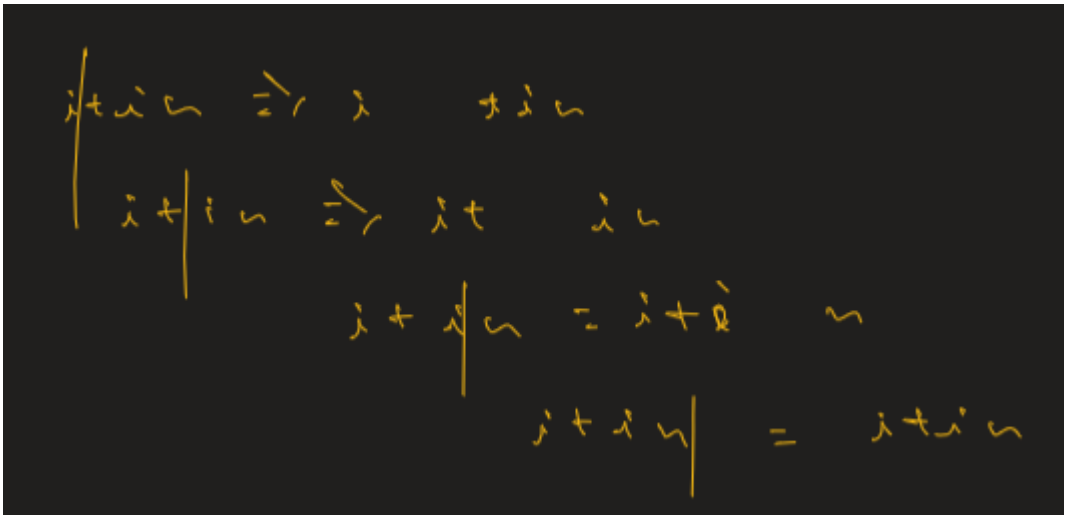
For example, if I split it after the first character **'n'**, I'll get two substrings:

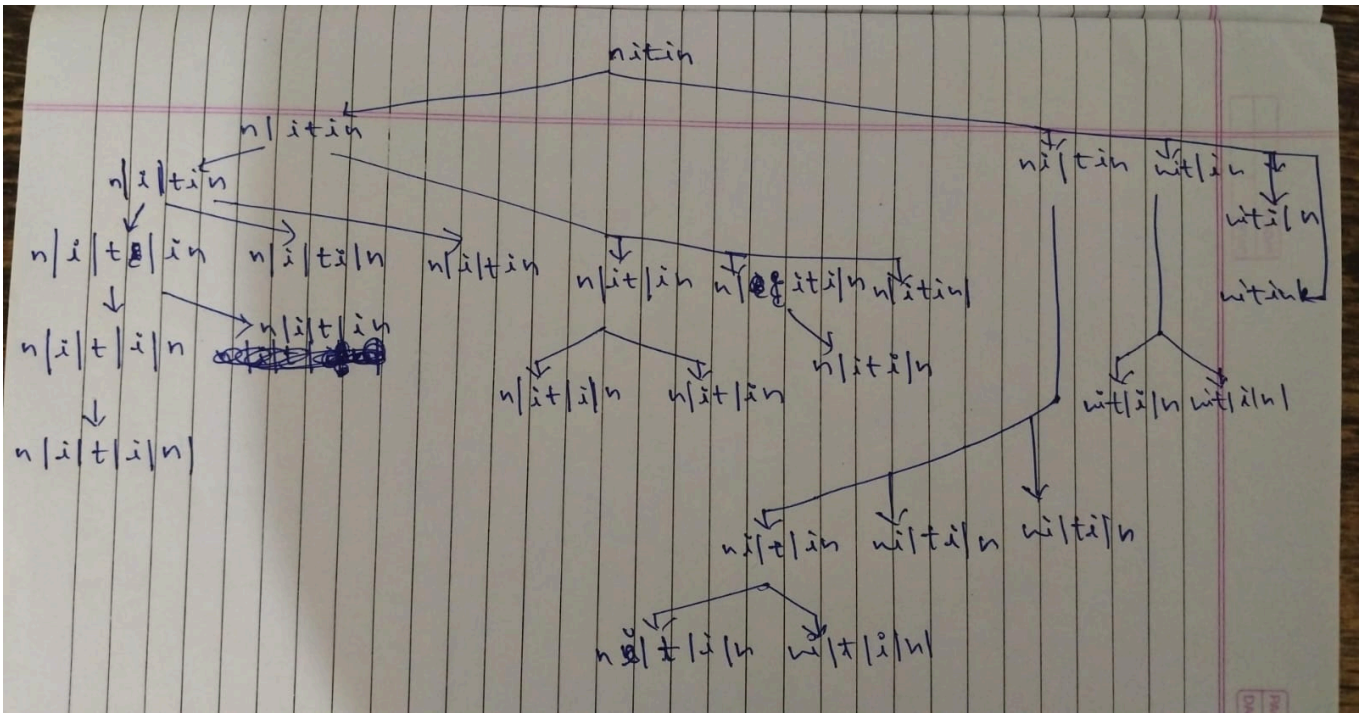
- Left: **"n"**
- Right: **"itin"**

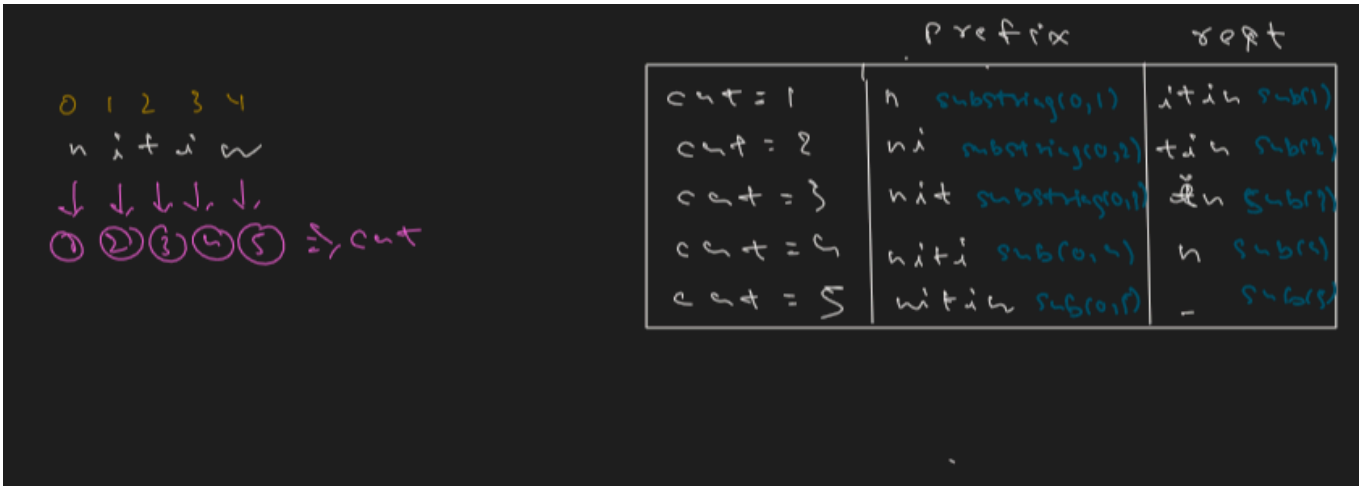
Next, I will recursively split the **right** substring (**"itin"**) into four parts, and continue this process until I reach individual characters.

Basically i will keep splitting the right part everytime










```java
public class Main {
    public static void main(String[] args) {
        String ip = "nitin";
        String op = "";
        partition(ip, op);
    }

    public static void partition(String ip, String op) {
```

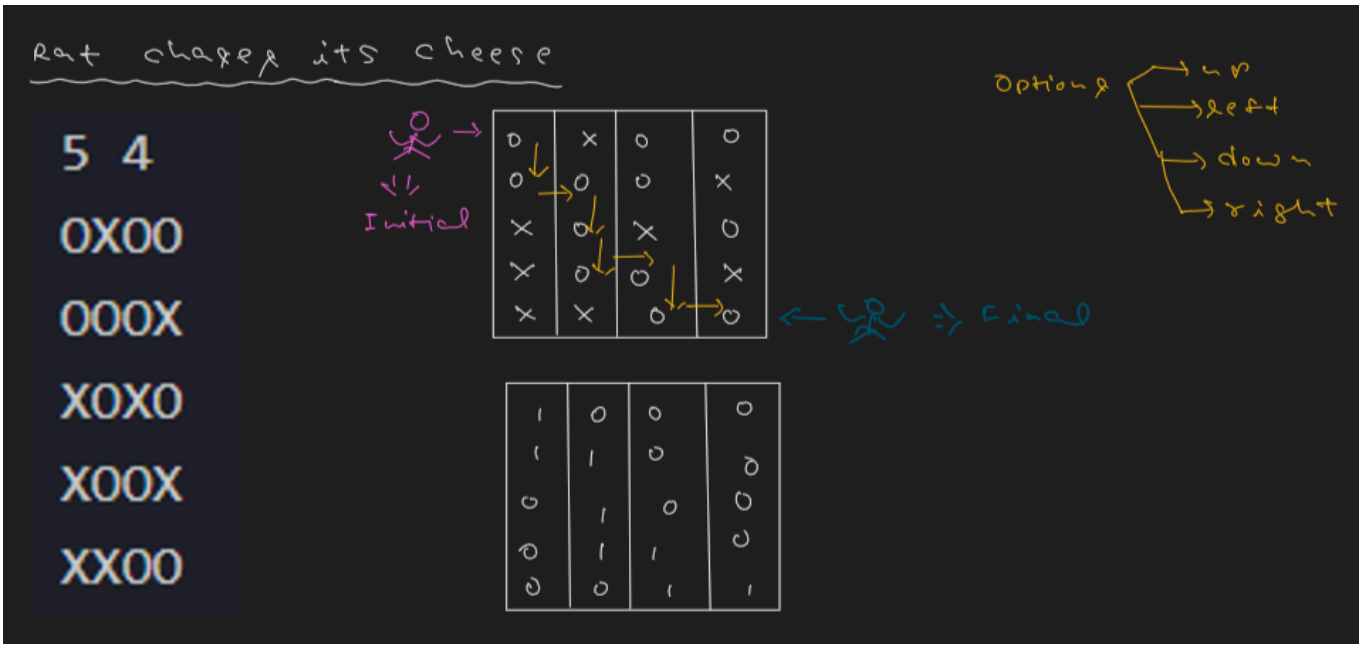
```
        if (ip.length() == 0) {
            System.out.println(op);
            return;
        }

        for (int cut = 1; cut <= ip.length(); cut++) {
            String prefix = ip.substring(0, cut);
            String rest = ip.substring(cut);
            partition(rest, op + prefix + "|");
        }
    }

}
```

```
public class Main {
      public static void main(String[] args) {
            String ip = "nitin";
            List<String> list = new ArrayList<>();
            List<List<String>> ans = new ArrayList<>();
            partition(ip, list, ans);
            System.out.println(ans);
      }

      public static void partition(String ip, List<String> list,
List<List<String>> ans) {
            if (ip.length() == 0) {
                  ans.add(list);
                  return;
            }

            for (int cut = 1; cut <= ip.length(); cut++) {
                  String prefix = ip.substring(0, cut);
                  if (isPalindrome(prefix)) {
                        List<String> tempList = new ArrayList<>(list);
                    tempList.add(prefix);

                    String rest = ip.substring(cut);
                          partition(rest, tempList, ans);
                  }
            }
      }

      public static boolean isPalindrome(String s) {
            int i = 0;
            int j = s.length() - 1;
            while (i < j) {
                  if (s.charAt(i) != s.charAt(j)) {
                        return false;
                  }
                  i++;
                  j--;
            }
            return true;
```

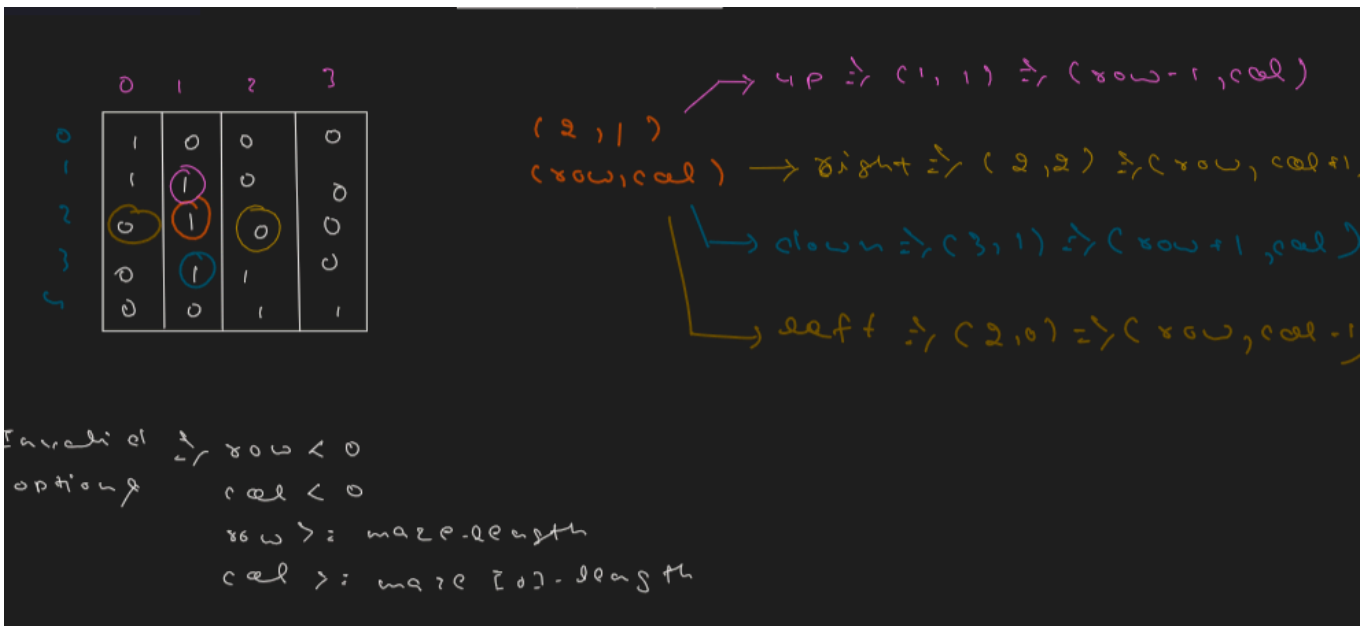
```
        }

}
```

```
class Solution {
    public List<List<String>> partition(String s) {
        String ip = s;
        List<String> list = new ArrayList<>();
        List<List<String>> ans = new ArrayList<>();
        partition(ip, list, ans);
        return ans;
    }

    public static void partition(String ip, List<String> list, List<List<String>> ans) {
        if (ip.length() == 0) {
            ans.add(list);
            return;
        }

        for (int cut = 1; cut <= ip.length(); cut++) {
            String prefix = ip.substring(0, cut);
            if (isPalindrome(prefix)) {
                List<String> tempList = new ArrayList<>(list);
                tempList.add(prefix);

                String rest = ip.substring(cut);
                partition(rest, tempList, ans);
            }
        }
    }

    public static boolean isPalindrome(String s) {
        int i = 0;
        int j = s.length() - 1;
        while (i < j) {
            if (s.charAt(i) != s.charAt(j)) {
                return false;
            }
            i++;
            j--;
        }
        return true;
    }
}
```

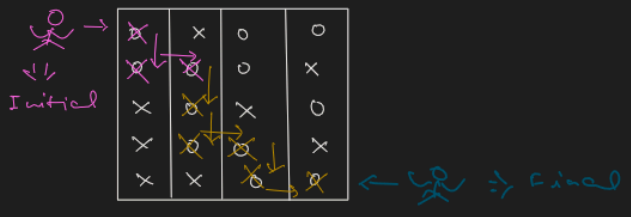
🎯 Rat Chases its cheese




```java
// Input
public class Main {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		int n = sc.nextInt();
		int m = sc.nextInt();
		char[][] maze = new char[n][m];

		for (int i = 0; i < maze.length; i++) {
			String s = sc.next();
			for (int j = 0; j < maze[0].length; j++) {
				maze[i][j] = s.charAt(j);
			}
		}
	}

}
```



```java
public static void printPath(char[][] maze, int row, int col){
    if(row < 0 || col < 0 || row >= maze.length || col >= maze[0].length){
        return;
    }
    printPath(maze, row-1, col); // up
    printPath(maze, row, col-1); // left
    printPath(maze, row+1, col); // down
    printPath(maze, row, col+1); // right
}
```




```java
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int m = sc.nextInt();
        char[][] maze = new char[n][m];
        for (int i = 0; i < maze.length; i++) {
            String s = sc.next();
            for (int j = 0; j < maze[0].length; j++) {
                maze[i][j] = s.charAt(j);
            }
        }
        int[][] ans = new int[n][m];
        printpath(maze, 0, 0, ans);
        if (!valueFound) {
            System.out.println("NO PATH FOUND");
        }

    }

    static boolean valueFound = false;

    public static void printpath(char[][] maze, int row, int col, int[][] ans) {
        if (row == maze.length - 1 && col == maze[0].length - 1 && maze[row][col] !=
'X') {
            ans[row][col] = 1;
            display(ans);
            valueFound = true;
            return;
        }
        if (row < 0 || col < 0 || row >= maze.length || col >= maze[0].length ||
maze[row][col] == 'X') {
            return;
        }
        maze[row][col] = 'X';
        ans[row][col] = 1;
        int[] r = { -1, 0, 1, 0 };
        int[] c = { 0, -1, 0, 1 };
        for (int i = 0; i < c.length; i++) {
            printpath(maze, row + r[i], col + c[i], ans);
        }
        // printpath(maze, row - 1, col, ans); // up
        // printpath(maze, row, col - 1, ans); // left
        // printpath(maze, row + 1, col, ans); // down
        // printpath(maze, row, col + 1, ans); // right
```

```
        maze[row][col] = 'O';
        ans[row][col] = 0;
    }

    public static void display(int[][] ans) {
        for (int i = 0; i < ans.length; i++) {
            for (int j = 0; j < ans[0].length; j++) {
                System.out.print(ans[i][j] + " ");
            }
            System.out.println();
        }

    }

}
```